Schema-on-read vs Schema-on-write

Suresh Podeti

3 min read · May 13, 2023

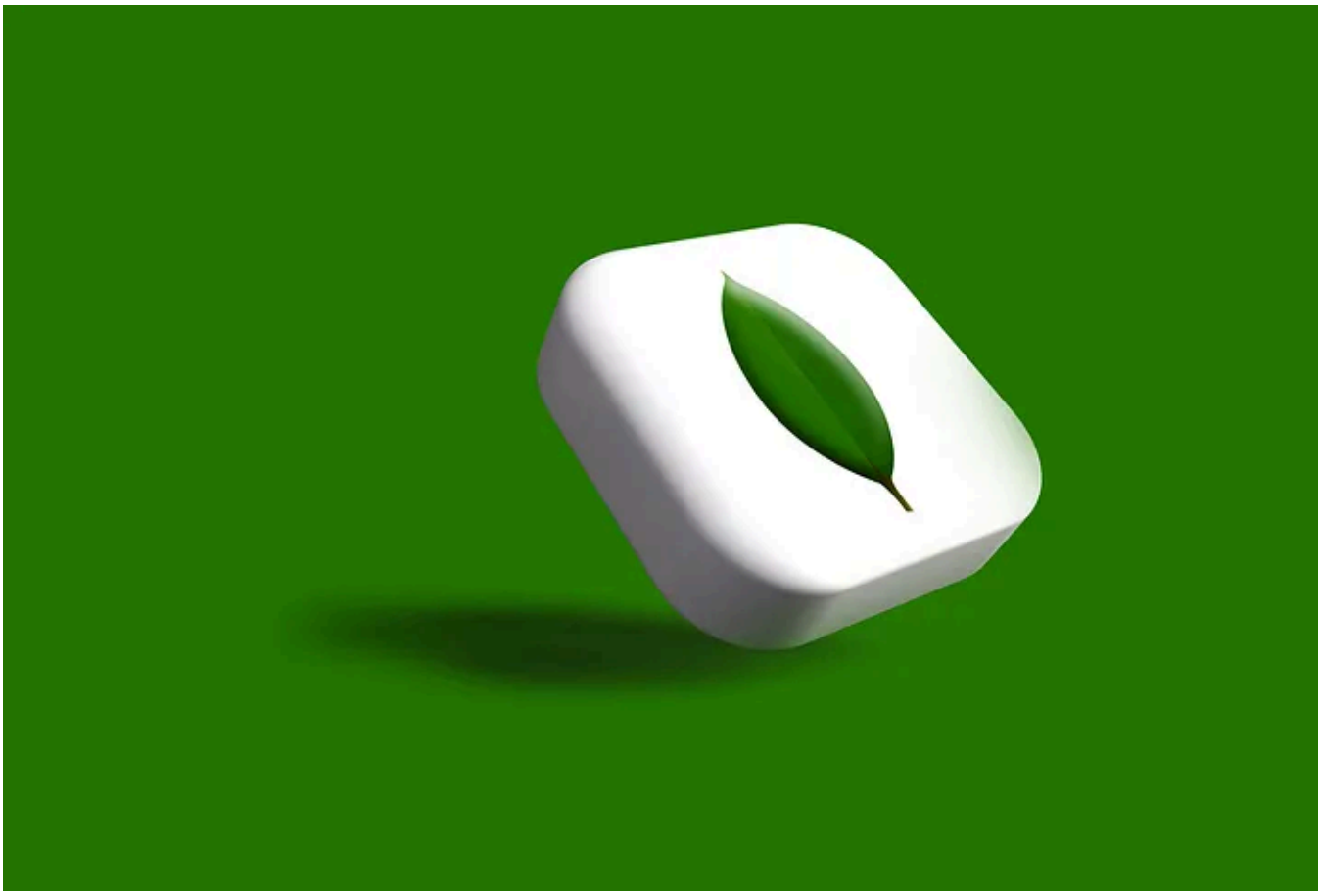

Photo by Rubaitul Azad on Unsplash

We have **two categories of databases** in general: 1. **Relational** 2. **Non-relational. Document-oriented** based databases are example of Non-relation databases eg. MongoDB, CouchDB, RethinkDB, etc..

Document databases are sometimes called ***schemaless,*** but that's **misleading**, as the code that **reads** the data usually assumes some kind of structures — i.e, there is an **implicit schema.** A more accurate term is **schema-on-read** (the structure of the data

is implicit, and only **interpreted when the data is read**), in contrast with **schema-on-write** (the traditional approach of **relational databases**, where the schema is explicit and the database ensures all written data conforms to it)**.**

Schema-on-read is similar to **dynamic (runtime) type checking** in programming languages, where as **schema-on-write** is similar to **static (compile-time) type checking .**

The difference between the approaches is particularly noticeable in situations where an application wants to change the format of its data. For example, say you are currently storing each **user's full name** in one filed, and you instead want to store the **first name and last name separately.** In a document database, you would just start writing new documents with the new fields and have code in the application that handles the case when old documents are read.

```
[
  {
    "id" 1,
    "full_name": "Suresh Podeti",
    "age": 28
  },
  {
    "id": 2,
    "full_name": "Nayan Podeti",
    "age": 1
  },
  {
    "id": 3,
    "first_name": "Navan",
    "last_name": "Podeti",
    "age": 1
  }
]
```

For example:

```
if(user && user.full_name && !user.first_name){
    full_name_split = user.full_name.split(" ");
     user.first_name = full_name_split[0];
    usr.last_name =full_name_split[1];
}
```

On the other hand, in a **statically typed** database schema, you would typically perform a migration along the lines of:

```
ALTER TABLE users ADD_COLUMN first_name text;
ALTER TABLE users ADD_COLUMN last_name text;

UPDATE users SET first_name=split_part(full_name, ' ',1); -- PostgreSQL
UPDATE users SET last_name=split_part(full_name, ' ', 2);-- PostgreSQL
```

Schema changes have a bad reputation of being slow and requiring downtime. Most relational database systems execute the ALTER TABLE statement in a few milliseconds. MySQL is a notable exception — it copies the entire table on ALTER TABLE, which can mean minutes or even hours of downtime when altering a large table — although various tools exist to work around this limitation.

Running UPDATE statement on large table is likely to be slow on any database, since every row needs to be rewritten.

The schema-on-read approach is advantageous if the items in the collection don't all have the same structure for some reason (i.e the data is heterogeneous), because:

1. There are many different types of objects, and it is not practical to put each type of object in its own table.
2. The structure of the data is determined by external systems over which you have no control and which may change at any time.

In situations like these, a schema may hurt more than it helps, and schema less documents can be much more natural data model. But in cases where all records are expected to have the same structure, schemas are a useful mechanism for documenting and enforcing that structure.

References:

1. O'reilly designing data-intensive applications by Martin Kleppmann, Chapter 2: Data models and Query languages.

Mongodb | Nosql Database | Schemaless | Schema On Read | Schema On Write

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium